Samantha Gregoryk
CPTS 315
November 23rd, 2020
Ananth Jillepalli

# Homework 4

**Question 1**

Suppose you are given the following multi-class classification training data, where each input example has three features and output label takes a value from *good, bad, and ugly*.

- $x_1$ = (0, 1, 0) and $y_1$ = *good*
- $x_2$ = (1, 0, 1) and $y_2$ = *bad*
- $x_3$ = (1, 1, 1) and $y_3$ = *ugly*
- $x_4$ = (1, 0, 0) and $y_4$ = *bad*
- $x_5$ = (0, 0, 1) and $y_5$ = *good*

Suppose we want to learn a linear classifier using multi-class perceptron algorithm and start from the following weights: $w_{good}$ = (0,0,0); $w_{bad}$ = (0,0,0); and $w_{ugly}$ = (0,0,0).

Please do hand calculations to show how weights change after processing examples in the same order (i.e., one single pass over the five training examples). See slide 88 of the Perceptron notes.

**$x_1$:**

$w_{good} = w_{good} + x_1$: (0,0,0) + (0,1,0) = (0,1,0) — $w_{good}$: (0,0,0)

$w_{bad} = w_{bad} - x_1$: (0,0,0) - (0,1,0) = (0,-1,0) — $w_{bad}$: (0,0,0)

$w_{ugly}$ = (0,0,0) — $w_{ugly}$: (0,0,0)

**$x_2$:**

$w_{good}$: (0,0,0) = (0,1,0) — $w_{good}$: (0, 1, 0)

$w_{bad}$: (0,0,0) = (0,-1,0) — $w_{bad}$: (0,-1, 0)

$w_{ugly}$: (0,0,0) = (0,1,0) — $w_{ugly}$: (0, 1, 0)

**$x_3$:**

$w_{good} = w_{good} \cdot x_3$: (0,1,0) · (1,1,1) = 1 — $w_{good}$: (0,0,0) = +1

$w_{bad} = w_{bad} \cdot x_3$: (0,-1,0) · (1,1,1) = -1 — $w_{bad}$: (0,0,0) = -1

$w_{ugly} = w_{ugly} \cdot x_3$: (0,0,0) · (1,1,1) = 0 — $w_{ugly}$: (0,0,0) = 0

$w_{good} = w_{good} - x_3$: (0,0,0) - (1,1,1) = (-1,0,-1) — $w_{good}$: (0,0,0) = (-1,0,-1)

$w_{bad}$ = (0,-1,0) — $w_{bad}$: (0,0,0) = (0,-1,0)

$w_{ugly} = w_{ugly} + x_3$: (0,0,0) + (1,1,1) = (1,1,1) $w_{ugly}$: (0,0,0) = (1,1,1)

**$x_4$:**

$w_{good} = w_{good} \cdot x_3$: (-1,0,-1) · (1,0,0) = -1 $w_{good}$: (0,0,0) = -1

$w_{bad} = w_{bad} \cdot x_3$: (0,-1,0) · (1,0,0) = 0 $w_{bad}$: (0,0,0) = 0

$w_{ugly} = w_{ugly} \cdot x_3$: (1,1,1) · (1,0,0) = 1 $w_{ugly}$: (0,0,0) = +1

$w_{good}$ =(-1,0,-1) $w_{good}$: (0,0,0) = (-1,0,-1)

$w_{bad} = w_{bad} - x_4$: (0,0,0) - (1,0,0) = (-1,0,0) $w_{bad}$: (0,0,0) = (-1,0,0)

$w_{ugly} = w_{ugly} + x_4$: (0,0,0) + (1,0,0) = (1,0,0) $w_{ugly}$: (0,0,0) = (1,0,0)

**$x_5$:**

$w_{good} = w_{good} \cdot x_5$: (-1,0,-1) · (0,0,1) = 1 $w_{good}$: (0,0,0) = +1

$w_{bad} = w_{bad} \cdot x_5$: (0,-1,0) · (0,0,1) = 0 $w_{bad}$: (0,0,0) = 0

$w_{ugly} = w_{ugly} \cdot x_5$: (1,1,1) · (0,0,1) = 1 $w_{ugly}$: (0,0,0) = +1

$w_{good} = w_{good} + x_5$: (0,0,0) + (0,0,1) = (0, 0, 1)

$w_{bad} = w_{good} - x_5$: (0,0,0) − (0,0,1) = (0, 0, -1)

$w_{ugly}$ = (1,0,0)

**$w_{good}$: (0, 0, 1)**

**$w_{bad}$: (0, 0, -1)**

**$w_{ugly}$: (1, 0, 0)**

**Question 2**

Suppose you are given the following binary classification training data, where each input example has three features and output label takes a value good or bad.

- $x_1$ = (0, 1, 0) and $y_1$ = *good*
- $x_2$ = (1, 0, 1) and $y_2$ = *bad*
- $x_3$ = (1, 1, 1) and $y_3$ = *good*
- $x_4$ = (1, 0, 0) and $y_4$ = *bad*
- $x_5$ = (0, 0, 1) and $y_5$ = *good*

Suppose we want to learn a classifier using kernelized perceptron algorithm. Start from the following dual weights: $a_1$ = 0; $a_2$ = 0; $a_3$ = 0; $a_4$ = 0; and $a_5$ = 0.

Please do hand calculations to show how dual weights change after processing examples in the same order (i.e., one single pass over the five training examples). Do this separately for the following kernels:

(a) Linear kernel: $K(x,\ x') = x * x'$; and (b) Polynomial kernel with degree 3: $K(x,\ x') = (x * x' + 1)^3$, where $x * x'$ stands for dot product between two inputs $x$ and $x'$.

See Algorithm 30 in http://ciml.info/dl/v0_99/ciml-v0_99-ch11.pdf. You can ignore the bias term b.

(a)

$\boldsymbol{x_1}$:

$$f(x) = \sum_{i=1}^{5} 0 \cdot \big((0,1,0) \cdot (0,1,0)\big) = 0 \cdot (0,1,0) = 0$$

$$a_1 = a_1 + 1$$
$$a_1 = 0 + 1$$
$$a_1 = 1$$

$$(x) = \sum_{i=1}^{5} 1 \cdot ((0,1,0) \cdot (1,0,1)) = 1 \cdot (0,0,0) = (0,0,0)$$

*a = [1,0,0,0,0]*

$\boldsymbol{x_2}$:

$$f(x) = \sum_{i=1}^{5} 0 \cdot \big((1,0,1) \cdot (1,0,1)\big) = 0 \cdot (1,0,1) = 0$$

*a = [1,0,0,0,0]*

$\boldsymbol{x_3}$:

$$f(x) = \sum_{i=1}^{5} 0 \cdot \big((1,1,1) \cdot (1,1,1)\big) = 0 \cdot (1,1,1) = 0$$

$$a_3 = a_3 + 1$$
$$a_3 = 0 + 1$$
$$a_3 = 1$$

$$(x) = \sum_{i=1}^{5} 1 \cdot \big((1,1,1) \cdot (1,0,0)\big) = 1 \cdot (1,0,0) = (1,0,0)$$

*a = [1,0,1,0,0]*

$\boldsymbol{x_4}$:

$$f(x) = \sum_{i=1}^{5} 0 \cdot \big((1,0,0) \cdot (1,0,0)\big) = 0 \cdot (1,0,0) = 0$$

*a = [1,0,1,0,0]*

$\boldsymbol{x_5}$**:**

$$f(x) = \sum_{i=1}^{5} 0 \cdot (0,0,1) \cdot (0,0,1) = 0 \cdot (0,0,1) = 0$$

$$a_5 = a_5 + 1$$
$$a_5 = 0 + 1$$
$$a_5 = 1$$

$$(x) = \sum_{i=1}^{5} 1 \cdot (1,1,1) \cdot (1,0,0) = 1 \cdot (1,0,0) = (1,0,0)$$

***a = [1,0,1,0,1]***

(b)

$\boldsymbol{x_1}$**:**

$$f(x) = \sum_{i=1}^{5} 0 \cdot \big((0,1,0) \cdot (0,1,0) + 1\big)^3 = 0 \cdot \big((0,1,0) + 1\big)^3 = 0$$

$$a_1 = a_1 + 1$$
$$a_1 = 0 + 1$$
$$a_1 = 1$$

$$f(x) = \sum_{i=1}^{5} 1 \cdot \big((0,1,0) \cdot (1,0,1) + 1\big)^3 = 1 \cdot \big((0,0,0) + 1\big)^3 = \big((0,0,0) + 1\big)^3$$

*a = [1,0,0,0,0]*

$\boldsymbol{x_2}$**:**

$$f(x) = \sum_{i=1}^{5} 0 \cdot \big((1,0,1) \cdot (1,0,1) + 1\big)^3 = 0 \cdot \big((1,0,1) + 1\big)^3 = 0$$

*a = [1,0,0,0,0]*

$\boldsymbol{x_3}$**:**

$$f(x) = \sum_{i=1}^{5} 0 \cdot \big((1,1,1) \cdot (1,1,1) + 1\big)^3 = 0 \cdot \big((1,1,1) + 1\big)^3 = 0$$

$$a_3 = a_3 + 1$$
$$a_3 = 0 + 1$$
$$a_3 = 1$$

$$(x) = \sum_{i=1}^{5} 1 \cdot \big((1,1,1) \cdot (1,0,0) + 1\big)^3 = 1 \cdot \big((1,0,0) + 1\big)^3 = \big((1,0,0) + 1\big)^3$$

*a = [1,0,1,0,0]*

$\boldsymbol{x_4}$**:**

$$f(x) = \sum_{i=1}^{5} 0 \cdot \big((1,0,0) \cdot (1,0,0) + 1\big)^3 = 0 \cdot \big((1,0,0) + 1\big)^3 = 0$$

*a = [1,0,1,0,0]*

$\boldsymbol{x_5}$**:**

$$f(x) = \sum_{i=1}^{5} 0 \cdot \big((0,0,1) \cdot (0,0,1) + 1\big)^3 = 0 \cdot \big((0,0,1) + 1\big)^3 = 0$$

$$a_5 = a_5 + 1$$
$$a_5 = 0 + 1$$
$$a_5 = 1$$

$$(x) = \sum_{i=1}^{5} 1 \cdot \big((1,1,1) \cdot (1,0,0) + 1\big)^3 = 1 \cdot \big((1,0,0) + 1\big)^3 = \big((1,0,0) + 1\big)^3$$

***a = [1,0,1,0,1]***

**Question 3**

Suppose $x = (x_1, x_2, \dots, x_d)$ and $z = (z_1, z_2, \dots, z_d)$ be any two points in a high-dimensional space (i.e., *d* is very large). Suppose you are given the following property, where the right-hand side quantity represents the standard Euclidean distance.

$$\left(\frac{1}{\sqrt{d}}\sum_{i=1}^{d} x_i - \frac{1}{\sqrt{d}}\sum_{i=1}^{d} z_i\right)^2 \leq \sum_{i=1}^{d}(x_i - z_i)^2$$

We know that the computation of nearest neighbors is very expensive in the high-dimensional space. Discuss how we can make use of the above property to make the nearest neighbors' computation efficient?

For both expressions on the left side of the inequality, they represent the projection of d-dimensions for points x and z. Each individual expression computes the average of the values. So, the squared distance for one dimensional projection for both x and z points will always be less than or equal to the distance between them in terms of d-dimension space (Euclidean distance). When calculating the mean of x and z projections and then later sorting the values, we can use this inequality to find their nearest neighbors.

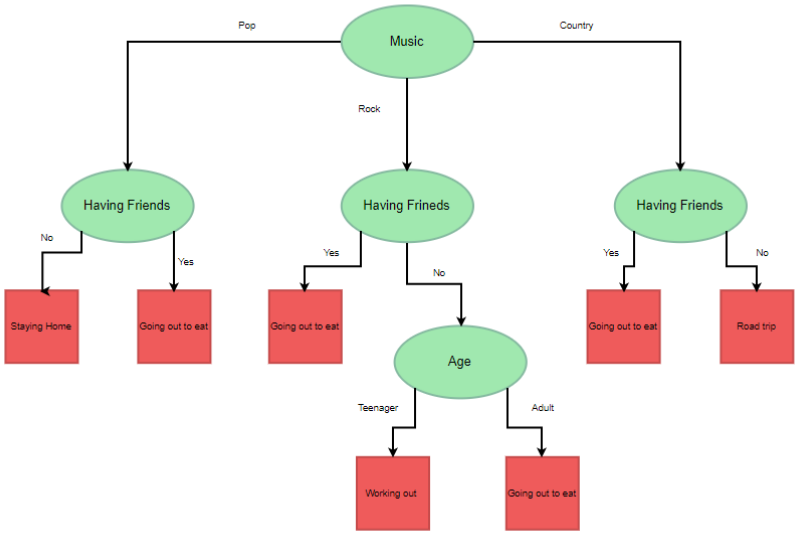
**Question 4**

We know that we can convert any decision tree into a set of if-then rules, where there is one rule per leaf node. Suppose you are given a set of rules $R = \{r_1, r_2, r_k\}$ where $r_i$ corresponds to the *i* th rule. Is it possible to convert the rule set R into an equivalent decision tree? Explain your construction or give a counterexample.

Yes, it is possible to convert the rule set R into an equivalent decision tree.

Example:

1. Having friends -> “Yes” & Music -> “Pop” & Age -> “Teenager” = Going out to eat
2. Having friends -> “No” & Music -> “Pop” & Age -> “Teenager” = Staying home
3. Having friends -> “No” & Music -> “Rock” & Age -> “Teenager” = Working out
4. Having friends -> “No” & Music -> “Rock” & Age -> “Adult” = Going out to eat
5. Having friends -> “No” & Music -> “Country” & Age -> “Adult” = Road trip




**Question 5**

Please read the following two papers and write a brief summary of the main points in at most THREE pages.

D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Franois Crespo, Dan Dennison: Hidden Technical Debt in Machine Learning Systems. NIPS 2015: 2503-2511 https://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems.pdf

Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, D. Sculley: The ML test score: A rubric for ML production readiness and technical debt reduction. BigData 2017: 1123-1132 https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/46555.pdf

*Hidden Technical Debt in Machine Learning Systems*

The production readiness and reduction of technical debt for ML system are found through testing and monitoring. Improving reliability, reducing technical debt, and lowering long-term maintenance cost can are also found through the technique of testing and monitoring as well. This paper covers the 28 actionable test to measure how ready for production a ML system is. Within each test, there is a written as an assertion; our recommendation is to test that the assertion is true, the more frequently the better, and to fix the system if the assertion is not true.

The first test is testing for model development. Since ML systems are not directly specified in code but is learned from data, this can cause issues that we need to make sure are clear for the system. The tests include; feature expectations are captured in a schema, all features are beneficial, no feature's cost is too much, features adhere to meta-level requirements, the data pipeline has appropriate privacy controls, new features can be added quickly, and all input feature code is tested.

The second test is testing for ML infrastructure. The best practices for ML models are still emerging in the ML model development world. Practices that are still in the development phase and need more time to be considered to be in the infrastructure are; training is reproducible, model specs are unit tested, the ML pipeline is Integration tested, model quality is validated before serving, the model is debuggable, models are canaried before serving and serving models can be rolled back.

The third test is monitoring tests for ML. Knowing if the ML system worked correctly at launch is vital but knowing that your ML system works correctly over time is even more important. Both attributes are just as important, and we must make sure that everything is running smoothly so we do not waste time and resources. These tests include; dependency changes result in notification, data invariants hold for inputs, training and serving are not skewed, models are not too stale, models are numerically stable, computing performance has not regressed and prediction quality has not regressed.

It is important to note the incentivizing culture change as well. A rubric was created to quantify a ML Test Score that can be measured and improved over time. This strategy was inspired by the Test Certified program at Google, which provided a scored ladder for overall test robustness, and which had strong success in incentivizing teams to adopt best practices. The ML Test Score is computed by; For each test, half a point is awarded for executing the test manually, with the results documented and distributed. A full point is awarded if there is a system in place to run that test automatically on a repeated basis. Sum

the score for each of the 4 sections individually. The final ML Test Score is computed by taking the minimum of the scores aggregated for each of the 4 sections.

Lastly, applying the rubric to real world systems covered unexpected insights. After they met with the 36 teams from Google that worked in a diverse array of product areas, they found that; there is an importance of checklists, be aware of dependency issues, there is an importance of frameworks and the need to assess the assessment.

*The ML test score: A rubric for ML production readiness and technical debt reduction*

The cost of ML system maintenance is experienced in every aspect of the business world. It is important to note the risk factors many companies experience and to take in account for a system design. These factors include boundary erosion, entanglement, hidden feedback loops, undeclared consumers, data dependencies, configuration issues, changes in the external world, and a variety of system-level anti-patterns.

Development of these systems can be expensive in the long run and should always be took into account. Companies that invest time and money into ML systems will succeed in the long run because of their wonderful capabilities. They need to make sure of their fiscal debt, technical debt, and hidden debt first. This paper's purpose is to increase the community's awareness of the difficult tradeoffs that must be considered in practice over the long term.

One practice is the complex models that erode boundaries. Traditional software engineering practice has shown that strong abstraction boundaries using encapsulation and modular design help create maintainable code in which it is easy to make isolated changes and improvements. To examine ways that the resulting erosion of boundaries may significantly increase technical debt in ML systems, we can look at entanglement, correction cascades, and undeclared consumers.

We must also be aware that data dependencies cost more than code dependencies because they carry a similar capacity for building debt, but may be more difficult to detect. It can be difficult to come back from building large data dependency chains. We must be on the look out for unstable data dependencies, underutilized data dependencies that can creep into a model through legacy features, bundled features, e-features and correlated features, and static analysis of data dependencies.

Detecting feedback loops can help ML systems since one of their key features is that they often influencing their own behavior if they update over time. Direct feedback loops and hidden feedback loops can be difficult to detect and address if they do occur over time in the ML system because some models are updated more frequently than others.

ML system anti patterns can be good to know when systems incorporate ML methods and end up with high-debt design patterns. These anti patterns include glue code, pipeline jungles, dead experimental code paths, abstraction debt and common smells (plain old data type smell, multiple language smell and prototype smell) and should be avoided at all costs when possible.

Configuration debt is another area where debt can accumulate is in the configuration of ML systems. Any large system has a wide range of configurable options, which includes which features are used, how data is selected, a wide variety of algorithm-specific learning settings, potential pre- or post-processing, verification methods, and so on. Being able to detect configurable options can save money in the long run since potential problems could be; serious loss of time, waste of computing resources, or production issues.

Dealing with change in the external world is one of the many things that makes ML so useful and popular in the business world. We all know how the external world is rarely stable and so it is important to be able to monitor and test ML systems at costs. This will improve our prediction bias, action limits, and up-stream producers to make ML systems better in the long run.